MapReduce Join Optimization

Jade Koskela

May 6, 2013

0

In this project I am implementing the MapReduce triple-equijoin algorithms described in *Optimizing Joins in a Map-Reduce Environment*. [1]

1

For a prototype 3-way join I used the Mondial dataset. I performed the join $country \bowtie ismember \bowtie organization$. The query corresponds to the SQL query:

```
SELECT
  code, abbreviation, c.name AS cname, NULL, o.name AS oname
  FROM country c, ismember m, organization o
  WHERE c.code=m.country AND m.organization=o.abbreviation;
```

The schema after projection performed in the mapper is:

```
country(code, name)
ismember(country, organization)
organization(abbreviation, name)
```

There are two sets of join keys, which in this case are both of size 1. The join key between country and ismember shall be referred to as A, and the join key between ismember and organization shall be referred to as B. Each join key has a key share, which represents the number of buckets the keys are hashed into. In this case both keys have a share of 5. There are 25 reducers

which represent a 5x5 grid. We can visualize this as follows:

		h(B)			
h(A)		X			

A row from the relation ismember has both join keys. In this case we know exactly which reducer to send the result to. In this case of the other relations, we must do a broadcast. A row from country will have to be multicast to all reducers in the associated row of the reducer matrix. Suppose we have a row from country, "F France Column3 Column4", and suppose that hash(F) = 3. We will construct 5 result key/value pairs which correspond to the 5 buckets of B, or columns of the matrix.

h(A)	X	X	X	X	X

Each key from the mapper is tagged with the relation tag. The relation tag for the middle relation is capitalized. This is to enforce an ordering on the keys as they are presented to the reducer. The reducer will process keys in the order: ismember, country, organization. In the reducer a hash join is performed. Each key from the middle relation, ismember, is inserted into the hash table with the key from country. Each row from country then joins

with rows from ismember and inserts the result into the hashtable using the key from organization. Each row from organization then completes the join and outputs the result.

2

I have implemented a more generalized 3 way join example, with the restriction that the relations are only 2 columns. I have also implemented a binary join example, which runs a pair of binary joins to perform the same operation. All of the join keys are projected out. Although this is an unlikely use case, the performance analysis will be the same. We can represent this as the following Datalog query:

```
Facts:
row(A,B)

Rules:
output(A,D) :- row(A,B), row(B,C), row(C,D)
```

I have compared the performance of the single step triple-join with the double-binary join. I have used the Facebook friend graph dataset. Each row corresponds to an edge in the graph. All the joins are self-joins. Since all relations are the same size, no reduce grid optimization was performed at this point. The reduce grid is a square matrix in each case. The results were as follows:

Rows	Reducers	Time Binary	Time Triple	Grid Size
1000	16	55s	23s	4x4
3000	16	63s	23s	4x4
10000	16	1m 11 s	1m 21s	4x4
10000	25	1m 33s	1m 21s	5x5
20000	25	1m 48s	4m 32s	5x5
20000	36	2m 23s	4m 1s	6x6

With the small datasets, triple-join finishes in half the time of binary-join. However this does not scale, which indicates that the performance on

the small dataset was dominated by the job setup time. As we increase the dataset size binary-join finishes in almost half the time of triple-join. The performance penalty incurred by triple-join is due to the intermediate data blowup. Since the left/right keys must be multicast to the reducers, the intermediate data is increased by O(n), where n is the size of the 'bucket set' of the other join key. In this case the size of the datasets are symmetric, so each grid is size nXn.

3

I have compared datasets of asymmetric sizes. This should allow for grid size optimization. The idea is that we can reduce the blowup of the intermediate data by allocating more buckets to the join key on the larger relation. The rows from the middle relation will only get sent to a single reducer, so no optimization can be performed there. In these examples, I have varied the size of the first relation, or the left relation in the three-way join. I have used the optimzation techniques outlined in [1], specifically the Lagrangian equation technique for finding the bucket sizes for minimum communication cost.

This 3-way join is expressed as $R(A, B) \bowtie S(B, C) \bowtie T(C, D)$. The bucket size or keyshare of the 1^{st} join key is represented as b, and the keyshare of the 2^{nd} is c. The size of the left relation R is represented as r, the size of T is represented as t. The product of the keyshares is equal to the number of reducers, or the size of our reducer matrix, k. Our communication cost equation is:

$$rc + tb - \lambda(bc - k)$$

We take the derivative with respect to each keyshare variable and set the resulting equations equal to 0.

$$r = \lambda b$$
$$t = \lambda c$$

We then multiply both sides by the missing share variable, keeping in mind that $bc = k$.

$$rc = \lambda k$$
$$tb = \lambda k$$

After solving this system of equations we get:

$$b = \sqrt{\frac{rk}{t}}$$
$$c = \sqrt{\frac{tk}{r}}$$

For testing the performance of my implementation I used a virtual cluster of size 20. This gives an approximate range for k, the number of reducers. I varied the size of the left dataset in the join. Since each row in the middle dataset is sent to only a single reducer, it does not have much of an effect on the communication cost. The size of the left and right side dataset of the join are listed in the table. To computed the optimized keyshare size, I assigned $r = 1$, and set t to a multiple of r, as reflected in the ratio of the relation sizes. The optimized grid sizes are then calculated and used for the following tests.

Rows	Reducers	Time Binary	Time Triple	Grid Size
20000,20000	25	2m 18s	1m 56s	5x5
10000,20000	24	2m 23s	1m 30s	3x8
5000,20000	20	1m 47s	1m 29s	2x10
50000,100000	24	2m 43s	2m 28s	3x8

The results were better than in the non-optimized self join cases. In all tests triple-join was faster than binary-join, but not by a large amount.

References

[1] Foto N. Afrati and Jeffrey D. Ullman. Optimizing joins in a map-reduce environment. In *Proceedings of the 13th International Conference on Extending Database Technology*, EDBT '10, pages 99–110, New York, NY, USA, 2010. ACM.